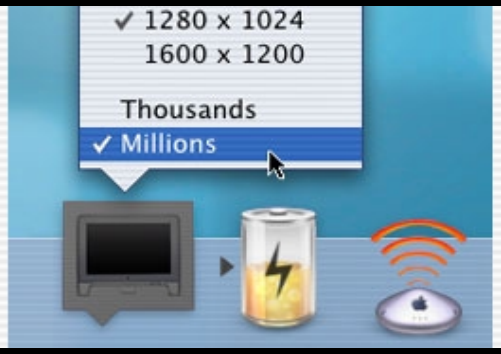
Session 311

Web Enabling Your Application



Richard Murphy
Manager, Security Platform Group

Introduction

- URL Access Overview
- URL Access APIs
- URL Access in Mac OS X 10.0
- nsURL/Cocoa web programming

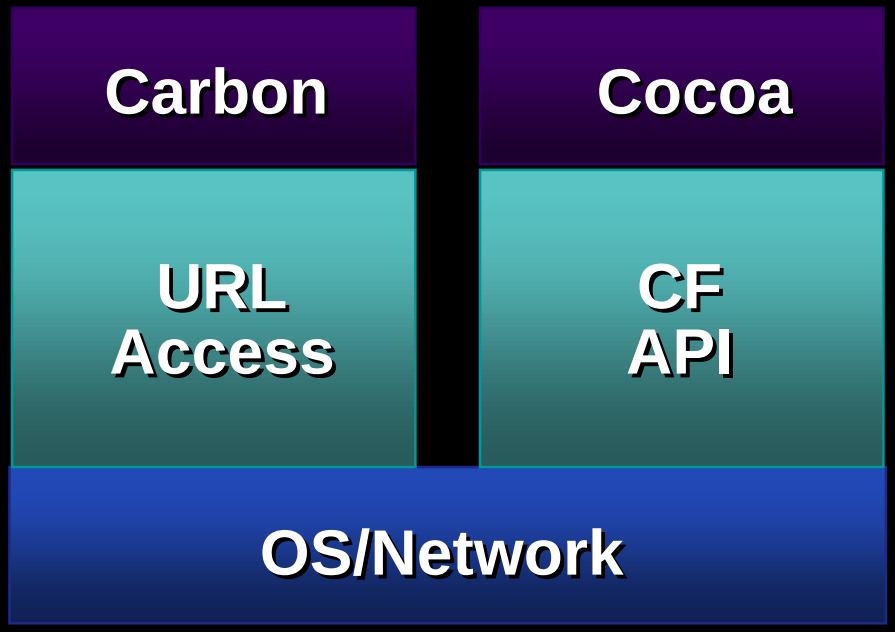
What You'll Learn

- What you can do with URL Access
- How to incorporate URL Access into your Carbon applications
- Limitations and future plans

What Is URL Access?

- An easy to use API to access data on the Internet
- Supports http://, https://, ftp://, and file://
- Downloading and uploading capabilities
- Carbon API available on Mac OS 9 and Mac OS X

Architecture



URL Access Features

General

- Authentication dialog
- Progress dialog
- File decoding
- Scriptable
- Proxy and firewall support
- Keychain support

URL Access Features

HTTP

- Download
- Post and put (requires some extra work)
- Auto-redirect
- Full access to server response
- Proxy authentication

URL Access Features

FTP

- Upload of files/directories
- Download of files/directories
- Directory listings

URL Access Features

HTTPS

- SSL 3.0
- 128-bit encryption

Functional Gotchas

- No HTML parsing
  - Redirection
  - File not found
  - Login
- Proxy limitations
  - FTP uploading
  - FTP directory downloading

Sample Uses

Third parties

- iCab uses it to download https sites
- Adobe uses it in Acrobat to implement "Web Capture"
- Numerous companies use scripting application internally

API Overview

- Synchronous API
  - System event procedure
- Flags
- Properties
  - HTTP Post
- Asynchronous API
  - Events
- States

Synchronous APIs

URLSimpleDownload

- Easiest way to download a URL
- You provide
  - Source URL
  - Destination FSSpec or handle
  - Flags indicating desired behavior
- Should be used in a thread

URLSimpleDownload

```
OSStatus GetURL(char* sourceURL, FSSpec* destination)
{
    URLOpenFlags        flags =
        kURLReplaceExistingFlag +
        kURLDisplayProgressFlag +
        kURLExpandFileFlag

    OSStatus status;
    status = URLSimpleDownload (
                        url,        // the url to download
                        destination,        // destination FSSpec
                        nil,        // don't download to memory
                        flags,      // control flags
                        nil,        // no system event  proc
                        nil);       // no user context
    return status;
}
```

System Event Procedure

- Parameter to synchronous calls
- Allows dialogs to be displayed as movable modal
- Update events will be passed through to proc
- Without one, dialogs will be non-movable modal

Flags

- Control optional aspects of the transfer
- Used in both synchronous and asynchronous calls

Sample Flags

- kURLReplaceExistingFlag
- kURLDisplayProgressFlag
- kURLDisplayAuthFlag
- kURLExpandFileFlag
- kURLBinHexFileFlag
- kURLNoAutoRedirectFlag

Synchronous APIs

URLDownload

- One extra step, significantly more powerful
- Must create and dispose a URLReference
  - URLNewReference, URLDisposeReference
- Should be used in a thread

Why Use URLDownload?

- Can get/set properties before and after download
- If threaded, you can
  - Monitor state of download
  - Abort download

URLDownload

```
OSStatus GetURL(char* sourceURL, FSSpec* destination)
{
    URLOpenFlags      flags = kURLReplaceExistingFlag;
    OSStatus status;
    URLReference urlRef;
    status = URLNewReference (sourceURL, &urlRef);
    status = URLDownload (
                        urlRef,  // the urlRef to download
                        destination,     // destination FSSpec
                        nil,     // don't download to memory
                        flags,   // control flags
                        nil,     // no system event  proc
                        nil);    // no user context
    URLDisposeReference (urlRef);
    return status;
}
```

Synchronous Uploads

- URLSimpleUpload, URLUpload
- Same as their download counterparts except:
  - Destination URL provided
  - Source FSSpec provided
  - Cannot upload from memory
  - Different flags apply
- FTP only

Properties

- Aspects of the URL
  - kURLHost
  - kURLURL
- Or the resource it points to
  - kURLResourceSize
  - kURLResourceName
  - kURLMIMEType

Properties (Cont.)

- Or the download
  - kURLStatusString
  - kURLTotalItems
- HTTP-specific
  - kURLHTTPRequestMethod
  - kURLHTTPRespHeaderBody
  - kURLHTTPRedirectedURL

Properties (Cont.)

- Retrieve with URLGetProperty
- Set with URLSetProperty
- May change during transfer

Settable Properties

- kURLRequestHeader
- kURLHTTPRequestMethod
- kURLHTTPRequestBody
- kURLHTTPUserAgent
- kURLUsername
- kURLPassword

HTTP Post

```
OSStatus DoPost(char* sourceURL, char* postData)
{
    OSStatus status;
    URLReference urlRef;
    status = URLNewReference (sourceURL, &urlRef);
    status = URLSetProperty (urlRef,
            kURLHTTPRequestMethod, “POST”, 4);
    status = URLSetProperty (urlRef,
            kURLHTTPRequestBody, postData,
            strlen(postData));
    status = URLDownload ();
    URLDisposeReference (urlRef);
    return status;
}
```

Asynchronous APIs

- Need a URLReference
- Call URLOpen
- Retrieve data with URLGetBuffer/URLReleaseBuffer
- Monitor using notification routine or URLGetCurrentState

Why Use Async APIs?

- If you are a filter
- If you want to display your own progress
- If you are anti-threading

Async Download

```
OSStatus GetURL(char* sourceURL, FSSpec* destination)
{
    URLOpenFlags        flags = kURLReplaceExistingFlag;
    URLReference urlRef;
    status = URLNewReference (sourceURL, &urlRef);
    URLNotifyUPP notifyUPP =
                        NewURLNotifyUPP (MyNotifier);
    status = URLOpen (urlRef,
                nil,        // don't download to file
                flags,      // control flags
                notifyUPP,          // notification routine
                kURLAllEventsMask, // register for all events
                &done);             // user context
    while (!done) URLIdle();
    DisposeURLNotifyUPP(notifyUPP);
    URLDisposeReference (urlRef);
    return status;
}
```

Notification Routine

```
OSStatus MyNotifier(void* userContext, URLEvent event,
                    URLCallbackInfo *callbackInfo)
{
    Boolean* done = (Boolean*) userContext;
    switch (event) {
        case (kURLDataAvailableEvent):
            ProcessData(callbackInfo->urlRef);
            break;
        case (kURLCompletedEvent):
        case (kURLErrorOccurredEvent):
            *done = true;
            break;
    }
    return noErr;
}
```

ProcessData

```
void ProcessData(URLReference urlRef)
{
    URLState currentState;
    void* buffer;
    Size bufferSize;
    do {
        URLGetCurrentState (urlRef, &currentState);
        URLGetBuffer (urlRef, &buffer, &bufferSize);
        …   // consume the buffer
        URLReleaseBuffer (urlRef, buffer);
    } while (currentState & kURLDataAvailableState);
}
```

Events

- Reported to notification routine
- Can register for subset only
  - kURLAllBufferEventsMask
  - kURLCompletedEventMask + kURLErrorOccurredEventMask

Sample Events

- kURLResourceFoundEvent
- kURLPropertyChangedEvent
- kURLErrorOccurredEvent
- kURLDataAvailableEvent
- kURLPeriodicEvent
- kURLAbortInitiatedMask

Async Download (Polling)

```
OSStatus GetURL(char* sourceURL, FSSpec* destination)
{
    URLReference urlRef;
    URLState currentState;
    status = URLNewReference (sourceURL, &urlRef);
    status = URLOpen (urlRef, destination, flags,
                      nil,      // no notification
                      0,        // no events
                      nil);     // no user context
    do {
        URLGetCurrentState(urlRef, &currentState);
        URLIdle();
    } while (!(state & kURLCompletedState) &&
                      !(state & kURLErrorOccurredEvent));
    URLDisposeReference (urlRef);
    return status;
}
```

States

- URLGetCurrentState reports state of transfer
- Some states have event counterparts
- Only thing you cannot find out with state is property changes
- Useful with both asynchronous and synchronous APIs

Sample States

- kURLConnectingState
- kURLDownloadingState
- kURLDataAvailableState
- kURLErrorOccurredState
- kURLCompletedState

API Review

- Synchronous API
- Asynchronous API
- Flags
- Properties
- Events
- States

Programming Gotchas

- URL Access is highly threaded!
  - Must call URLIdle
- SetProperty expects c/Pascal strings (not standardized)
- Cannot call URLDisposeReference from notification routine

Programming Gotchas (Cont.)

- HTTP Request body cannot contain null bytes (<= OS 9.0.4)
- URLReference is not reusable
- Authentication dialog not supported with URLOpen

Mac OS 9 vs. Mac OS X

- Plug-ins not supported on Mac OS X
- Properties work “correctly” on Mac OS X
- File protocol not available on Mac OS X

Getting Started

- To develop on Mac OS 9
  - Download Security SDK—ftp://ftp.apple.com/developer/Development_Kits/Security_SDK.sit.hqx
  - Or CarbonLib 1.0.2 or later—ftp://ftp.apple.com/developer/Development Kits/CarbonLib_1.3.1GM_SDK.img.bin

Getting Started (Cont.)

- To develop on Mac OS X
  - Install Mac OS X
  - Link with /System/Library/Frameworks/Carbon.Framework
  - Latest documentation—
    http://developer.apple.com/techpubs/Carbon/networkcomm/URLAccessManager/urlaccessmanager.html

Future Plans (Mac OS X)

- New URL Access Implementation targeted to use Mac OS X system features
- Less threads in the library
- More stable
- Streaming upload/download capability

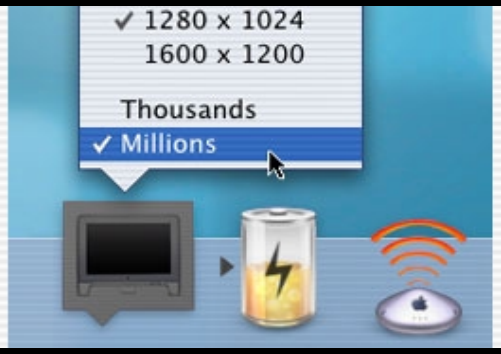
Future Plans (Mac OS X)

- Connection reuse (sessions)
- SecureTransport library to use security Framework root certs

Summary

- URL Access allows you to access Internet data
- Carbon API
- Easy to use synchronous APIs
- Flexible asynchronous APIs

Web Access From Cocoa



Becky Willrich
Core Frameworks

CFURL

- Simple CFType used to represent URLs
- Is bridged to NSURL
- Parses the URL string and gives you access to its components
- Interprets percent-escapes for you
  - Assumes UTF-8 where encodings aren't specified
- Is a pure data class
  - Doesn't do any file or network I/O

Representing Files as CFURLs

- Use the functions with “File” in their names
  - CFURLCreateWithFileSystemPath()
  - CFURLCreateFromFileSystemRepresentation()
  - CFURLCreateWithFileSystemPathRelativeToBase()
  - CFURLCreateFromFileSystemRepresentationRelativeToBase()
- Can also convert between FSRefs and CFURLs
  - CFURLCreateFromFSRef()
  - CFURLGetFSRef()

Web Access From Cocoa

- Using URL Access
- Using NSURL
- Using NSURLHandle
- Extending NSURLHandle

URL Access From Cocoa

- Use URL Access as you would from a Carbon application
- Don't use any of the UI switches
  - Carbon panels and Cocoa panels cannot co-exist
- URL Access uses ThreadManager threads, so you must be ThreadManager conscious

URL Access From Cocoa– The ThreadManager

- You must call URL Access from a ThreadManager thread
  - The main thread is O.K.
  - The main thread must yield
- If you use URLOpen(), you must use URLIdle()
  - Best to post a notification with posting style NSPostWhenIdle, then call URLIdle() when you receive the notification

NSURL—Synchronous

- Synchronous
  - **NSData *urlData = [myURL resourceDataUsingCache:YES];**
- Cache flag allows you to keep the URL's data cached for later (re)access

NSURL—Asynchronous

- Start the asynchronous download
  - **[myURL loadResourceDataNotifyingClient:self usingCache:YES];**
- Implement the NSURLClient informal protocol on your client object
  - **- (void)URL:(NSURL *)sender resourceDataDidBecomeAvailable:(NSData *)newBytes;**
  - **- (void)URLResourceDidFinishLoading:(NSURL *)sender;**
  - **- (void)URLResourceDidCancelLoading:(NSURL *)sender;**
  - **- (void)URL:(NSURL *)sender resourceDidFailLoadingWithReason:(NSString *)reason;**

What's Really Going On?

- NSURL never does the download directly
- NSURLHandle manages the download
- When NSURL starts a download, it builds a connection to an NSURLHandle

Using the NSURLHandle

- You can obtain the NSURLHandle directly
  - `NSURLHandle *handle = [myURL URLHandleUsingCache:YES];`
- Manipulating it gives you more direct control
- You can also use it to flush the cache
- Look in NSURLHandle.h for details

What's Supported?

- **http:** and **file:** only
- No proxy or firewall support
- No **https:** support
- The good news:
  - These are all future directions
  - The architecture is extensible

Extending NSURL(Handle)

- You can implement new schemes or override existing ones by subclassing NSURLHandle
- NSURLHandle was designed for subclassability
- Once your subclass is registered, NSURL will use it automatically

Subclassing NSURLHandle

- Create a subclass
  - **@interface MySubclass:NSURLHandle**
- Implement all methods marked as "to be overridden" in NSURLHandle.h
- Register your subclass with NSURLHandle
  - **[NSURLHandle registerURLHandleClass:MySubclass];**

Methods to Override

```
+ (BOOL)canInitWithURL:(NSURL *)anURL;
+ (NSURLHandle *)cachedHandleForURL:(NSURL *)anURL;
- initWithURL:(NSURL *)anURL cached:(BOOL)willCache;
- (id)propertyForKey:(NSString *)propertyKey;
- (id)propertyForKeyIfAvailable:(NSString *)propertyKey;
- (BOOL)writeProperty:(id)propertyValue forKey:(NSString *)propertyKey;
- (BOOL)writeData:(NSData *)data;
- (NSData *)loadInForeground;
- (void)beginLoadInBackground;
- (void)endLoadInBackground;
```

Methods to Use

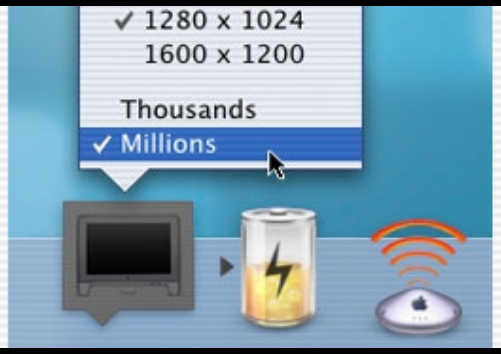
```
- (void) backgroundLoadDidFailWithReason:
   (NSString *)reason;

- (void)
   didLoadBytes:(NSData *)newBytes
   loadComplete:(BOOL)yorn;
```

Who to Contact

Thomas Weyer
Network & Communications Technology Manager
Apple Worldwide Developer Relations
weyer@apple.com

http://developer.apple.com/events/urls.html

Q&A



Thomas Weyer
Network & Communications Technology Manager
weyer@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001